```
Trying 01082...Open

PLEASE ENTER HOST PORT ID:
PLEASE ENTER HOST PORT ID:x
LOGINID:d232mbg
PASSWORD:
TERMINAL (ENTER 1, 2, 3, 4, OR ?):□3

  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  *                                                                 *
  *           Welcome to MESSENGER (APS Text) at USPTO              *
  *                                                                 *
  *                                                                 *
  *    The USPTO production files are current through:              *
  *    MAR 24 1998 for U.S. Patent Text Data.                       *
  *    MAR 24 1998 for U.S. Current Classification data.            *
  *    MAR 24 1998 for U.S. Patent Image Data.                      *
  *                                                                 *
  *       * PLEASE USE 305-9000 FOR NEW TELEPHONE NUMBER *          *
  *                                                                 *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * More U.S. patent data is now available on APS.  The new        *
  * USOCR file contains patents issued in 1970, plus some          *
  * patents that were missing from the USPAT file.  See the        *
  * Patents News Folder under the Public Folders in e-mail for     *
  * more information on using the new file.  Thank you.            *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  *  DISCLAIMER:                                                    *
  *  Neither the United States Government, nor any agency           *
  *  thereof, nor any of their contractors, subcontractors or       *
  *  employees make any warranty, expressed or implied,             *
  *  including any warranty of marketability of fitness for a       *
  *  particular purpose; nor assumes any legal liability or         *
  *  responsibility for any party's use, or the results of          *
  *  such, of the data.                                             *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  *                Help Desk --> 703-305-9000                       *
  *                                                                 *
  *    The Help Desk is staffed for APS support 7 days/week.        *
  *      Monday through Friday:      6:30am - 9:00pm                *
  *      Saturday, Sunday, Holidays: 8:30am - 5:00 pm               *
  *                                                                 *
  *    The Help Desk staff at this number will handle all APS       *
  *    related questions.                                           *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  *    >>>>>>>>>>>>> NEW SUNDAY HOURS !!! <<<<<<<<<<<<<             *
  *                                                                 *
  *    The APS is available:                                        *
  *           6:30am - 9:00pm Monday through Friday                 *
  *           7:30am - 5:00pm Saturday, Sunday, Holidays            *
  *                                                                 *
  *      APS is unavailable Thanksgiving Day, Christmas Day,        *
  *      and New Year's Day.                                        *
  *                                                                 *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
FILE 'USPAT' ENTERED AT 11:02:55 ON 31 MAR 1998

  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
string on the P-ATM.
//If the client needs to retrieve data from upstream ho
(OmniHost, VISA etc), or needs data from a database it makes
a TCP stream connection to a servlet.
//This is consistent with the behavior of the network
terminal which would  //make the same connection over PPP.
clienTransObject = new Socket (<Host>, <Well known
socket>);
//Explained further down under initial client requests
. . . . .   //Further processing
//Send out host requests
clienTransObject.write (HostRequest);
clienTransObject.read (HostResponse);
IOObject.WriteString (StringIndex + n);.  .  .
```

DETDESC:

DETD(50)

Initial Client Request Processing

DETDESC:

DETD(51)

The AD runs a T thread (spawned off during startup) that listens on a well-known socket (e.g. 1112) waiting for **initial** ClientRequests from a **client** application. The T thread processes the ClientRequest to determine which **servlet** class needs loading.

DETDESC:

DETD(52)

```
{
ClientInitialRequestListener = new ServerSocket (<wellknown
socket (e.g. 1112)>);
//Wait for initial requests and spawn off server
connSocket = ClientInitialRequestListener.accept( );
connSocket.Stream.read (InitialRequest);
Parse (InitialRequest);
HostThread HO = new Thread (connSocket, "class name");
}
```

DETDESC:

DETD(53)

The T thread is a daemon thread and lives as long as the AD lives. When the **client** application is **downloaded** to a Java capable terminal **initial** requests arrive over the PPP link.

DETDESC:

DETD(55)

Host Threads (H) **service client** requests for upstream and database **connectivity**. A host thread can make TCP **connections** with remote hosts, forward financial transactions originating from the **client** application and route the response.

DETDESC:

DETD(59)

Consistence in the a[redacted]ss of transport layer services [redacted]eds no over emphasis. The design o[redacted]the PTS **server** aims to provide [redacted]uniform interface to third party **client** component and **server** component **applet** writers to the async dial-up protocol module and the system's TCP/SLIP/PPP stack. This interface comprises a set of Java Classes. . .

DETDESC:

DETD(61)

A uniform upper edge interface for the ModemIO classes permits easy replacement of the implementation. The actual modem handling **code**, for instance, may use the TAPI **client** calls instead of direct Win32 communication calls. Multiple libraries that conform to the same interface allow different link level protocol. . .

DETDESC:

DETD(63)

1. Open an end-to-end async, duplex dial-up **connection**. The station address (InetAddress as in TCP/IP) is the dial string. **Configure** upon **connection**.

DETDESC:

DETD(64)

2. Listen for an incoming dial-up **connection**. The listen port (analogous to the bound TCP port) is the COM port. In this regard the valid port numbers range from 0-0xFF (which is the maximum number of COM ports allowed in NT). **Configure** upon **initialization**.

DETDESC:

DETD(65)

3. Obtain Input and Output streams that re-direct from/to the open **connection**.

DETDESC:

DETD(66)

4. Hang-up (close as in TCP/IP) a live **connection**.

DETDESC:

DETD(70)

Modem **Initialization** And Methods

DETDESC:

DETD(71)

. . .

```
sInitString);
public ModemPort (Port objPort, String sInitString);
public ModemPort (String sPortName, String sInitString);
//Methods
public Port getPort( );
public boolean connect (String sDialString);
public void disconnect( );
public void reset( );
public boolean configure (String sCfgStr);
```

public boolean **configureDM** (String sCfgStr);
}

DETDESC:

DETD(72)

Programmers must use getPort() to capture a stream and transfer data over the ModemPort. **Configure**(String) sends out an AT command and returns TRUE if the modem returned OK<cr><lf>. **configureDM**(String) sends out the same command to the modem when in data mode.

DETDESC:

DETD(74)

The . . . interface and yet avoid conflicts, the package instantiates its own socket and serversocket objects via constructors that take an extra **parameter** (that **identifies** the lower object that needs to be instantiated). This is illustrated after the class definition.

DETDESC:

DETD(76)

The . . . that corresponds to the machines TCP/IP address. The async dial-up line may however use multiple COM ports to open a **connection** with the host. Heuristically, it may seem that fitting the TCP/IP host/machine address into the native COM support library will. . . the host name (as a String) will, instead, be overloaded. This value will now refer to a dial String that **identifies** the remote station address.

DETDESC:

DETD(77)

Socket **Initialization** And **Connection**

DETDESC:

DETD(84)
BOOL **ConfigureDevice** (HANDLE hDev, DCB * pNewDCB);

CLAIMS:

CLMS(1)

Having . . . what we claim as new, and desire to secure by letters patent is:

1. A distributed computer system including a **client** terminal and a **server** which communicate via a network, comprising:
(a) the **client** terminal **initiating connection** to the **server** utilizing the network;
(b) the **server** responding to the **initial connection** by transmitting an enquiry message to the **client** terminal;
(c) the **client** terminal responding to the enquiry message with a message comprising **identification** information indicative of the **client** terminal being a network terminal or a non-network terminal and **identifying** a **client** application the **client** terminal requires;
(d) the **server** receiving and analyzing the **identification** information to determine if the **client** terminal is a network terminal or a non-network terminal; and
(d1) if the **client** terminal is a network terminal, then the

client loader on the **server** starts the **client** application, if necessary, on the **client** terminal utilizing the network and starts a **server** application to **service** future requests from the **client** terminal; and

(d2) if the **client** terminal is a non-network terminal, then the **server initiates** the **client** application and **server** application on the **server** for processing the **client** application at the **server** for the **client** terminal.

CLAIMS:

CLMS(2)

2. The distributed computer system as recited in claim 1, wherein the start of the **client** application entails a, **download** of the **client** application to the **client** terminal.

CLAIMS:

CLMS(3)

3. The distributed computer system as recited in claim 1, in which the **client** terminal communicates to the **server** utilizing a dial-up network **connection**.

CLAIMS:

CLMS(4)

4. The distributed computer system as recited in claim 1. wherein the **identification** information comprises **configuration** characteristics of the **client** terminal.

CLAIMS:

CLMS(5)

5. The distributed computer system as recited in claim 1, wherein the network terminal executes Java **code** on the network terminal.

CLAIMS:

CLMS(6)

6. The distributed computer system as recited in claim 1, wherein the same **client** application is executed on the **server** and the **client** terminal.

CLAIMS:

CLMS(7)

7. The distributed computer system as recited in claim 1, wherein the non-network terminal receives commands from the **client** application on the **server**.

CLAIMS:

CLMS(8)

8. The distributed computer system as recited in claim 1, including means for passing a **client** application request to another **server** to process the request.

CLAIMS:

CLMS(9)

9. A method for distributing computing between a **server** and a **client** terminal which communicate via a network, comprising the steps of:

(a) **initiating connection** of the **client** terminal to the **server** utilizing the network;

(b) responding to the **initial connection** request at the **server** by transmitting an enquiry message to the **client** terminal;

(c) responding to the enquiry message at the **client** terminal with a message comprising **identification** information indicative of the **client** terminal being a network terminal or a non-network terminal and **identifying** a **client** application the **client** terminal requires;

(d) receiving and analyzing the **identification** information at the **server** to determine if the **client** terminal is a network terminal or a non-network terminal; and

(d1) loading a **server** application if the **client** terminal is a network terminal, which starts the **client** application if necessary, on the **client** terminal utilizing the network and starts a **server** application to **service** future requests from the **client** terminal; and

(d2) loading a **server** application on the **server**, if necessary, which **initiates** a **client** application on the **server** for processing the **client** application at the **server** on behalf of the **client** terminal, if the **client** terminal is a non-network terminal.

CLAIMS:

CLMS(10)

10. The method as recited in claim 9, wherein the start of the **client** application entails a **download** of the **client** application to the **client** terminal.

CLAIMS:

CLMS(11)

11. The method as recited in claim 9, including the step of communicating between the **client** terminal and the **server** utilizing a dial-up network **connection**.

CLAIMS:

CLMS(12)

12. The method as recited in claim 9, wherein the **identification** information comprises **configuration** characteristics of the **client** terminal.

CLAIMS:

CLMS(13)

13. The method as recited in claim 9, wherein the network terminal executes Java **code** on the network terminal.

CLAIMS:

CLMS(14)

14. The method as recited in claim 9, wherein the same **client** application is executed on the **server** and the **client** terminal.

CLAIMS:

CLMS(15)

15. The method as recited in claim 9, wherein the non-network terminal receives commands from the **client** application on the **server**.

CLAIMS:

CLMS(16)

16. The method as recited in claim 9, including the step of passing a **client** application request to another **server** to process the request.

CLAIMS:

CLMS(17)

17. A computer program embodied on a computer-readable medium for enabling a distributed computing system, including a **client** terminal and a **server** which communicate via a network, comprising:

(a) a **code** segment for **initiating connection** of the **client** terminal to the **server** utilizing the network;

(b) a **code** segment for responding to the **initial connection** request at the **server** by transmitting an enquiry message to the **client** terminal;

(c) a **code** segment for responding to the enquiry message at the **client** terminal with a message comprising **identification** information indicative of the **client** terminal being a network terminal or a non-network terminal and **identifying** a **client** application the **client** terminal requires;

(d) a **code** segment for receiving and analyzing the **identification** information at the **server** computer to determine if the **client** terminal is a network terminal or a non-network terminal; and

(d1) a **code** segment for loading a **server** application if the **client** terminal is a network terminal, which starts the **client** application if necessary, on the **client** terminal utilizing the network and starts a **server** application to **service** future requests from the **client** terminal; and

(d2) a **code** segment for loading a **server** application, if necessary, on the **server** which **initiates** the **client** application on the **server** for processing the **client** application at the **server** on behalf of the **client** terminal, if the **client** terminal is a non-network terminal.

CLAIMS:

CLMS(18)

18. The computer program as recited in claim 17, wherein the start of the **client** application entails a **download** of the **client** application to the **client** terminal.

CLAIMS:

CLMS(19)

19. The computer program as recited in claim 17, including a **code** segment for communicating between the **client** terminal and the **server** utilizing a dial-up network **connection**.

CLAIMS:

CLMS(20)

20. The computer program as recited in claim 17, wherein the **identification** information comprises **configuration** characteristics of the **client** terminal.

CLAIMS:

CLMS(21)

21. The computer program as recited in claim 17, wherein the network terminal executes Java **code** on the network terminal.

CLAIMS:

CLMS(22)

22. The computer program as recited in claim 17, wherein the same **client** application is executed on the **server** and the **client** terminal.

CLAIMS:

CLMS(23)

23. The computer program as recited in claim 17, wherein the non-network terminal receives commands from the **client** application on the **server**.

CLAIMS:

CLMS(24)

24. The computer program as recited in claim 17, including a **code** segment for passing a **client** application request to another **server** to process the request.

CLAIMS:

CLMS(25)

25. The computer program as recited in claim 17, including a **code** segment for making a dial up **connection** appear to the **server** as a socket **connection**.

=> d his

```
     (FILE 'USPAT' ENTERED AT 06:46:32 ON 06 APR 1999)
L1             1 S SERVLET AND SERVICE
L2             1 S L1 AND (PARAM? OR CONFIG? OR DOWNLOAD? OR SERVER OR CLIE
NT
L3             1 S L2 AND (CONNECT? OR INIT?)
L4             1 S L3 AND (IDENTIF? OR ID OR ACCOUNT OR CODE OR LOGIN OR PA
SSW
L5             1 S L4 AND PROXY
```

terminal session to a **user** of the **client** system. The emulator is operable to **download** [redacted] **applet code** to a **client** syste[redacted] in response to receiving a uniform resource locator associated with the legacy host system.

DETDESC:

DETD(9)

Client . . . to communicate with a legacy host system across a persistent TCP/IP socket connection 30. Client thread 28 is operable to **interface** between TCP/IP environment 14 and public Internet/intranet environment 16. **Applet** process 42 executes under web browser 38 and is operable to communicate with client thread 28 across persistent TCP/IP socket connection 44. In one embodiment of the present invention, **client** thread 28 and **applet** process 42 using a web/emulator data flow. **Applet** process 42 is further operable to provide a terminal session to a **user** of the client system 36. The terminal session can comprise a 3270, 5250, NVT or VT220 type terminal session. This terminal session provided by **applet** process 42 and client thread 28 is supported by a persistent TCP/IP socket connection which allows real-time time bidirectional communication. . .

DETDESC:

DETD(10)

In one embodiment of the present invention, the **applet** process uses three threads to implement the terminal session. The **applet** process uses a READ thread, a WRITE thread and an EMULATION thread. The READ thread provides a buffer **interface** to persistent TCP/IP socket connection 44 and reads information from the socket. The WRITE thread provides a buffer **interface** to persistent TCP/IP socket connection 44 and writes to the socket. The EMULATION thread provides the **user interface** representing the terminal session on the legacy host system. In this embodiment, the **applet** process and the **client** thread communicate using a web/emulator data flow based upon the communication protocol that is set forth in APPENDIX A.

DETDESC:

DETD(12)

The . . . 18 and UNIX host system 19 through web browser 38 without the need for specialized emulation software to be manually **loaded** on each **client** system 36 and without the need for **user** programming. Web/emulator **server** 26 **downloads applet code** 34 to **client** system 36 when the uniform resource locator (URL) of the legacy host system is selected by the **user**. This is accomplished automatically without requiring **service** personnel to **load** software on **client** system 36.

DETDESC:

DETD(19)

The . . . as 3270, 5250, NVT and VT220 type terminal sessions. According to the present invention, terminal emulation is partially provided by **applet** executable **code downloaded** from the web/emulation **server**. The **user** can select the uniform resource locator (URL) of the legacy host system via a web browser package, such as NETSCAPE's NETSCAPE NAVIGATOR, and transparently receive the **applet code** which is executed and invokes an appropriate terminal session.

CLAIMS:

CLMS(1)

What . . .

legacy host system, comprising:
a client thread operable to communicate with a legacy host system across a first persistent connection; and
applet code operable to create an **applet** process executing under a web browser on a **client** system, the **applet** process operable to communicate with the **client** thread across a second persistent connection and to provide a terminal session to a **user** of the **client** system;
the **server** operable to **download** the **applet code** to a **client** system in response to receiving a uniform resource locator associated with the legacy host system;
wherein the client thread is operable to communicate using a TN3270 data flow and the **applet** process is operable to provide a 3270 type terminal session.

CLAIMS:

CLMS(6)

6. . . . legacy host system, comprising:
a client thread operable to communicate with a legacy host system across a first persistent connection; and
applet code operable to create an **applet** process executing under a web browser on a **client** system, the **applet** process operable to communicate with the **client** thread across a second persistent connection and to provide a terminal session to a **user** of the **client** system;
the **server** operable to **download** the **applet code** to a **client** system in response to receiving a uniform resource locator associated with the legacy host system;
wherein the client thread is operable to communicate using a TN5250 data flow and the **applet** process is operable to provide a 5250 type terminal session.

CLAIMS:

CLMS(7)

7. . . . legacy host system, comprising:
a client thread operable to communicate with a legacy host system across a first persistent connection; and
applet code operable to create an **applet** process executing under a web browser on a **client** system, the **applet** process operable to communicate with the **client** thread across a second persistent connection and to provide a terminal session to a **user** of the **client** system;
the **server** operable to **download** the **applet code** to a **client** system in response to receiving a uniform resource locator associated with the legacy host system;
wherein the client thread is operable to communicate using a NVT data flow and the **applet** process is operable to provide a NVT type terminal session.

CLAIMS:

CLMS(8)

8. . . . legacy host system, comprising:
a client thread operable to communicate with a legacy host system across

a first persistent connection; and
applet code operable to create an **applet** process executing under a web browser on a **client** system, the **applet** process operable to communicate with the **client** thread across a second persistent connection and to provide a terminal session to a **user** of the **client** system;
the **server** operable to **download** the **applet code** to a **client** system in response to receiving a uniform resource locator associated with the legacy host system;
wherein the client thread is operable to communicate using an VT220 data flow and the **applet** process is operable to provide a VT220 type terminal session.

US PAT NO: 5,742,768 [IMAGE AVAILABLE] L19: 20 of 25

ABSTRACT:
A method for providing a web page (26) having an embedded menu (46) to a web browser (24) and for displaying the web page (40) to a **user** of the web browser (24) are provided. A request for a web page (20) is received from a web browser (24). In response to the **request**, a web page (26) and an **applet** (28) associated with the web page (20) are packaged for transmission to the web browser (24). The web page (26) and the **applet** (28) are then transmitted to and **downloaded** by the web browser (24). When the web page (26) is displayed and the **applet** (28) is executed by the web browser (24), the **applet** (28) creates and manages an embedded menu (46) in the displayed web page (40)under control of the **applet** (28). This embedded menu (46) provides a user of the web browser (24) with a plurality of links (48) through one action in the displayed web page (40).

ABSTRACT:
A . . . (26) having an embedded menu (46) to a web browser (24) and for displaying the web page (40) to a **user** of the web browser (24) are provided. A request for a web page (20) is received from a web browser (24). In response to the **request**, a web page (26) and an **applet** (28) associated with the web page (20) are packaged for transmission to the web browser (24). The web page (26) and the **applet** (28) are then transmitted to and **downloaded** by the web browser (24). When the web page (26) is displayed and the **applet** (28) is executed by the web browser (24), the **applet** (28) creates and manages an embedded menu (46) in the displayed web page (40)under control of the **applet** (28). This embedded menu (46) provides a user of the web browser (24) with a plurality of links (48) through. . .

SUMMARY:

BSUM(11)

According to another aspect of the present invention, a method for displaying a web page having an embedded menu to a **user** of a web browser is provided. This method includes **downloading** a web page and an **applet** from a web **server**. The web page is then displayed to a **user** of the web browser, and the **applet** is executed by the web browser. An embedded menu is created and managed in the displayed web page under control of the **applet**. This embedded menu provides a user of the web browser with a plurality of links through one action in the. . .

DETDESC:

DETD(4)

In . . . 18 can receive the request from web browser 24 and, in response, can package and transmit web page 26 and **applet** 28 to web browser 24 across IP network 16. Web server 18 packages web page 26 and

applet 28 based upon web page 20 and **applet** 22 stored on host system 12. After **downl[redacted]ng** web page 26 and **applet** 28, [redacted] browser 24 can display [redacted]b page 26 to a **user** of **client** s[redacted]tem 14 and can execute **applet** 28. **Applet** 28 only needs to be **downloaded** once and is executed by web browser 24. When a link is selected in web page 26, web server 18. . .

DETDESC:

DETD(7)

Display window 30 includes a displayed web page, indicated generally at 40, which is generated by web browser 24 from the **downloaded** web page 26 and associated **applet** 28. Displayed web page 40 provides the **user** of web browser 24 with the information content accessed from web **server** 18. The **user** generally interacts with display window 30 and displayed web page 40 using a pointer device (e.g., a mouse) which controls the position of a pointer 42 and allows a **user** to initiate actions (e.g., through a mouse click). According to the teachings of the present invention, displayed web page 40 includes a plurality of hot spots 44 that provide access to embedded menus created and managed by **applet** 28. The embedded menus can be accessed by positioning pointer 42 over one of hot spots 44.

CLAIMS:

CLMS(9)

9. A method for displaying a web page having an embedded menu to a **user** of a web browser, the method comprising:
downloading a web page and an **applet** transmitted by a web **server**;
displaying the web page to a **user** of the web browser; and
executing the **applet**, the **applet** creating and managing an embedded menu in the displayed web page under control of the **applet**, the embedded menu providing a user of the web browser with a plurality of links through one action in the. . .

US PAT NO: 5,729,594 [IMAGE AVAILABLE] L19: 21 of 25

ABSTRACT:
A remote communication system for facilitating secure electronic purchases of goods in on-line, wherein a suitable local user input device in association with a data transmission system, couples the user input into a packet network system for communication to a remote receiver/decoder apparatus to TRY a potentially desired product. Upon selection of the desired product by the user, a telcom network link is used to communicate a telephone number associated with the desired product from the user to the remote receiver to allow the user to BUY the desired product. The telcom network used to link the user input device to the remote apparatus may also include a 900 number billing system for assessing and collecting fees for use of the system.

DETDESC:

DETD(63)

The . . . Window event processing allows the plug-in to respond to mouse clicks in the plug-in window. In FIG. 9, assume the **user** clicks on the ESC button 122. This button should close the Netscape Navigator window 120, cancel the operations, and return to the Netscape Navigator. Assume the **user** clicks on the TRY button 124. The action to be taken will depend upon the form of the product information. . . the simplest case, a URL 126 allows the Netscape Navigator to retrieve and display the product information. Alternatively, a JAVA **applet** may be

downloaded and executed, or an application-to-application procedure may be executed by the plug-in "TRY" process. Thus, the possible responses to selections. . .

US PAT NO: 5,706,502 [IMAGE AVAILABLE] L19: 22 of 25

ABSTRACT:
A portfolio management system (PMS) is disclosed that allows users to manage, create, edit, debug and compile software portfolios that can include several different types of components, or projects. For example, projects can be Java applets, standalone executable programs, image files, Java class libraries or remote Java applets. The software portfolios and/or their constituent projects can be stored on the system hosting the portfolio management system or on any remote system that can be accessed via the Internet using standard Internet communications protocols, such as FTP or HTTP. The PMS includes portfolio files, each of which includes links to the projects that compose a portfolio and project files that set out the attributes of one project. The PMS also provides portfolio methods that allow users to create, choose, import and remove entire portfolios and project methods that allow users to create, import, choose, edit, remove, run, copy and paste projects. The contents of a particular portfolio or project file determines how the PMS implements the aforementioned methods. For example, if a user wants to import a portfolio from a remote system, the PMS invokes an integrated Web browser, which downloads the desired portfolio onto the local system. The PMS also allows users to publish portfolios and projects on the Internet to be used by others within certain limits set by the publisher. For example, the publisher can restrict copying of source programs while allowing copying of executables.

SUMMARY:

BSUM(8)

Important . . . result, Java language programs cannot create object pointers and generally cannot access system resources other than those resources which the **user** explicitly grants it permission to use. Consequently, when one or more **code** fragments are downloaded to a **client** along with an associated form or image file, a Java-compatible browser running on the client will be able to verify and execute the **downloaded** code fragments needed to display the image or fill out the forms. However, no Java-compatible browser provides any kind of programming environment; instead, browsers are strictly directed to end users browsing existing HTML files and using the referenced Java **applets** and image files.

DETDESC:

DETD(3)

Details . . . an application called the Java Workshop (JWS) program 150A, which, among other things, allows users to organize executable programs (Java **applets** and standalone executables) and non-executable files (image files and Java **class** libraries) into collections called portfolios. In a major departure from the prior art in the area of **program** and file managers, the JWS **program** 150A has an integrated JWS Browser 154A that allows a **user** seamlessly to create and work with portfolios that are **remote** (stored apart from the **user**'s machine or local network) or local. Furthermore, the JWS browser 154A allows portfolios to be assembled that are mixtures of local and **remote** "projects".

DETDESC:

DETD(5)

The **user interface** of the present invention facilitates **user** interaction with mixed objects, such as a portfolio consisting of both local and **remote** projects, by providing a single paradigm for working with all objects regardless of the objects' locations. Of course, there are differences between working with remote and local objects. For example, executing a Java **applet** stored on a **remote** computer is a vastly different task from executing a standalone **program** stored locally. These differences are handled in the JWS **program** 150A. However, the **user interface** of the present invention allows a **user** to initiate the execution of the **remote applet** or the local **program** in the same way (e.g, by double-clicking an icon representing the **applet**). How the **user interface** of the present invention provides this location-transparency flexibility is now described in reference to FIG. 1.

DETDESC:

DETD(8)

Each . . . neither will this application. As with the links 118Ai, the references 124Ai in the Web documents 120A can be to **remote** or local files. In either case, they are handled by the JWS browser 154A in the same manner as described for the links 118Ai. One significant advantage of the present **user interface** is that a reference 124Ai can be to a Java **applet** 140Ai that is responsible for handling the operations associated with the icon IAi whose related Web document referenced the **applet** 140Ai. In this situation, when the JWS browser 154A retrieves the web document 120Ai linked to a selected icon IAi,. . .

DETDESC:

DETD(25)

For . . . there is shown a portfolio file 160A1 that contains project file references 162A1j for its constituent projects, which include an "**Applet**", a "Standalone" **program**, a Java "Package", an "Image" and a "**Remote**" **applet**, all of which are local projects stored in the **user**'s "home" (i.e., local) directory in the memory 106A. Because these projects are all stored in the **user**'s "home" directory, they can be read and written by the **user** and their corresponding project files can be referenced by path and file name. For example, the reference 162A1a to the **applet** project file 170A1 is "/home/**Applet**.prj". The portfolio file 160A1 also includes a reference 162A1f (/lib/SemiRemote.prj) to a project file for a read-only project ("SemiRemote") stored. . .

DETDESC:

DETD(41)

A **user** can create a Java **applet** project, a standalone **program** project, a Java package project, an image project or a **remote** project. To create any of these projects, the **user** first "Chooses" the portfolio with which the project is to be associated and selects the "Create" option from the Project. . .

DETDESC:

DETD(42)

In some situations (when the project being created is an **applet**, standalone **program** or Java package) the **user** may also have access to source **code** for the newly-created project. In these situations,

the **user** enters the file names of the corresponding source files on the Project->Create pag[redacted]47A2a. The JWS 150A adds these [redacted]ource. . . file names to a "Source[redacted] list maintained in the memory [redacted]6A so the source files can be accessed by the **user** (e.g., for editing and compilation). The **user** also enters the name of the main file for the **program** (i.e., the file that contains the "main" routine) of which the newly created project is a part. When the project being created is a Java **applet**, it is possible that the Java **applet** is referenced in an HTML page so that, when the **applet**'s reference is selected, the **applet** will be executed. The JWS 150A allows such relationships to be represented via a Run Page URL field in the. . .

DETDESC:

DETD(50)

A **user** can then run the **remote applet** by selecting its name from the Project->Run submenu 147A2f or by **loading** the Portfolio Manager, selecting the **remote** project and then pressing the Run button IA7 on the toolbar 160. The Project->Run method 146A2f then passes the URL of the Web page referenced in the run page URL field (e.g., 184A5) of the remote **applet** project file (170A5) to the Web browser 154A, which **downloads** the referenced Web page (http://C.com/RunApplet2.htm) and runs the remote **applet** (**Applet2**).

CLAIMS:

CLMS(19)

19. . . .
attributes including a project type name and location, said project type being selected from a predefined set of values including **applet** and at least one of standalone **program**, Java package, image file and remote **applet**;
writing said attributes to a new project file having a project file name and location that are derived from said project name and location;
when said project type is selected from said **applet**, standalone **program** or Java package values:
enabling said **user** to enter source **code** links to source **code** files associated with said new project; and
writing said source **code** links to said new project file; and
writing into a particular portfolio file associated with a particular portfolio of which. . .

CLAIMS:

CLMS(20)

20. . . . of claim 19, further comprising the steps of:
when said project type is entered by said user as said remote **applet** value, enabling said **user** to enter a project URL **identifying** a remote **applet** and a run page URL **identifying** the URL of an HTML page that includes an **applet** tag referencing said **remote applet**; and
writing said project URL and run page URL to said new project file;
so that, when a **user** runs said new project using said run method, said HTML page **identified** by said run page URL is **downloaded** to and executed by said local computer, which, under control of said executing HTML page, **downloads** and runs said **remote applet**.

US PAT NO: 5,691,897 [IMAGE AVAILABLE] L19: 23 of 25

ABSTRACT:
A system for motion control in which an application is developed that is

independent from the actual motion control hardware used to implement the system. The system comprises a software system that employs an application programming interface comprising component functions and a service provider interface comprising driver functions. A system programmer writes an application that calls the component functions. Code associated with the component functions relates these functions to the driver functions. A hardware designer writes driver code that implements the driver functions on a given motion control hardware product. The driver functions are separated into core and extended driver functions. All software drivers implement the core driver functions, while the software drivers need not contain code for implementing the extended driver functions. If the software driver does not contain code to implement an extended driver function, the functionality of the extended driver function is obtained through a combination of core driver functions. The system programmer may also select one or more streams that allow the control commands to be communicated to, and response data to be communicated from, motion control hardware.

DETDESC:

DETD(36)

Referring again for a moment to FIG. 1, this Figure illustrates that the system 22 additionally comprises a driver administrator CPL **applet** 38 and a DDE **server** 40. The driver administration CPL **applet** 38 generates the **user interface** through which the **user** 24 communicates with the driver administrator module 32. The DDE **server** 40 provides the software **interface** through which the application **program** 26 communicates with the motion control component module 35.

DETDESC:

DETD(157)

The driver administrator CPL module 38 provides the **user-interface** that allows the **user** to add, configure, and remove drivers and streams in the system 22. The driver administrator 32 handles all driver and stream management. Even though the control panel **applet** provides the **user-interface**, this module 32 does the actual management work.

DETDESC:

DETD(198)

The following discussion describes the module interaction map for the control panel **applet** 38. A module is defined as either an executable binary, an external data file, or a main **user-interface** element used when interacting with the **user**. FIG. 51 is a drawing of all modules that interact with each other when running the driver administrator control panel **applet**.

US PAT NO: 5,572,648 [IMAGE AVAILABLE] L19: 24 of 25

ABSTRACT:
Method and apparatus for changing a dynamic tool palette in accordance with a current context of an application includes providing a static display of windowing functions and a dynamic display of windowing functions wherein the dynamic display is altered in accordance with a current context of an application. In the invention, the application is executed and the context of the application is registered with a context memory. The registered context is examined and a determination is made as to the applicable windowing functions associated with the registered context. In accordance with the determination result, the dynamic display is modified.

DETDESC:

DETD(7)

In operation, upon selecting the folder **applet**, the **applet** for that function is **downloaded** from a file server and stored in a random access memory (RAM) from where it can be executed. After the folder **applet** is **downloaded** from the file **server**, its context which includes a **program** handle and database **identification** number is registered with a context manager **applet** 21. In the present invention, context manager 21 resides as part of the static tool palette display **applet** 20. However, context manager 21 may also be a separate **applet** which operates throughout the present **program**.

DETDESC:

DETD(23)

Upon . . . both the static and dynamic tool palettes are displayed on monitor 2 in step S900. In step S901, a container **applet** is selected in order to create a container for holding data. Upon selecting a container **applet**, the container **applet** is **downloaded** from the file **server** and stored in a RAM location for execution. The context of the container **applet** which contains the container's handle and database **identification** is registered with context manager 21 in step S902. After the container context has been registered and the container created, an editor **applet** is selected and executed in step S903. At this point, the editor **applet** examines the context of the current container which is stored in context manager 21. In step S903, the editor applet. . .

US PAT NO: 5,548,745 [IMAGE AVAILABLE] L19: 25 of 25

ABSTRACT:
A method and apparatus for defining a context environment of an editor applet which is processing data to be stored in a container applet. In the invention, a first applet is executed and the context environment indicative of the first applet is stored. A second applet is executed and examines the stored context environment indicative of the first applet. In accordance with the result of the examination, the second applet modifies its operation.

DETDESC:

DETD(7)

In operation, upon selecting the folder **applet**, the **applet** for that function is **downloaded** from a file server and stored in a random access memory (RAM) from where it can be executed. After the folder **applet** is **downloaded** from the file **server**, its context which includes a **program** handle and database **identification** number is registered with a context manager **applet** 21 shown in FIG. 6. In the present invention, context manager 21 resides as part of the static tool palette display **applet** 20. However, context manager 21 may also be a separate **applet** which operates throughout the present **program**.

DETDESC:

DETD(23)

Upon . . . both the static and dynamic tool palettes are displayed on monitor 2 in step S900. In step S901, a container **applet** is selected in order to create a container for holding data. Upon selecting a container **applet**, the container **applet** is **downloaded** from the

(3) **Downloads** an **applet** (e.g., answering machine software) to new node 100 and installs it in the appropriate directory.

DETDESC:

DETD(824)

New . . . the remote confidence test, new node 100 sends CTS 5104 an appropriate acknowledgement/failure message (step 5308). CTS 5104 may then **download** a free **applet** onto new node 100 (step 5310), which installs the **applet** (step 5312). A purpose of the free **applet** is to encourage new users to register their nodes. After CTS 5104 disconnects (step 5314), CTS 5104 deposits the registration. . .

DETDESC:

DETD(839)

After . . . confidence test was passed and the CTS.sub.-- Status variable is set to the value CTS.sub.-- PASSED (step 5516). The free **applet** is then **downloaded** to the new node 100 and a message is sent to instruct the new node 100 to hang up (step. . .

=> d his

(FILE 'USPAT' ENTERED AT 11:02:55 ON 31 MAR 1998)

```
L1            19 S APPLET
L2             1 S CONFIGURATION(30A)L1
L3            11 S DOWNLOAD?(30A)L1
```

```
*          W E L C O M E   T O   T H E                    *
*        U. S.   P A T E N T   T E X T   F I L E          *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

=> s applet and servlet

            19 APPLET
             0 SERVLET
L1           0 APPLET AND SERVLET

=> s java

L2         126 JAVA

=> s javascript

L3           1 JAVASCRIPT

=> d

1.  5,572,643, Nov. 5, 1996, Web browser with dynamic display of
information objects during linking; David H. Judson, 395/200.48; 379/89;
707/531 [IMAGE AVAILABLE]

=> d ab

US PAT NO:      5,572,643 [IMAGE AVAILABLE]              L3: 1 of 1

ABSTRACT:
A method of browsing the Worldwide Web of the Internet using an
HTML-compliant client supporting a graphical user interface and a
browser. The method begins as a web page is being displayed on the
graphical user interface, the web page having at least one link to a
hypertext document preferably located at a remote server. In response to
the user clicking on the link, the link is activated by the browser to
thereby request downloading of the hypertext document from the remote
server to the graphical user interface of the client. While the client
waits for a reply and/or as the hypertext document is being downloaded,
the browser displays one or more different types of informational
messages to the user. Such messages include, for example, advertisements,
notices, messages, copyright information and the like.
```